Classical realizability in the CPS target language

Frey, Jonas

Published in:
Electronic Notes in Theoretical Computer Science

DOI:
10.1016/j.entcs.2016.09.034

Publication date:
2016

Document version
Publisher's PDF, also known as Version of record

Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 325 (2016) 111–126

www.elsevier.com/locate/entcs

Classical Realizability in the CPS Target Language

Jonas Frey[1]

Department of Computer Science
University of Copenhagen, Denmark
jofr@di.ku.dk

Abstract

Motivated by considerations about Krivine's classical realizability, we introduce a term calculus for an intuitionistic logic with record types, which we call the *CPS target language*. We give a reformulation of the constructions of classical realizability in this language, using the categorical techniques of realizability triposes and toposes.
We argue that the presentation of classical realizability in the CPS target language simplifies calculations in realizability toposes, in particular it admits a nice presentation of conjunction as intersection type which is inspired by Girard's ludics.

Keywords: Classical realizability, ludics, topos, tripos, CPS translation.

1 Introduction

The relationship between *continuation passing style (CPS) translations* of the λ-calculus, *negative translations* of classical into intuitionistic logic, *control operators* in abstract machines, and *evaluation order* (*call-by-value* vs. *call-by-name*) was uncovered during the 70's, 80's, and early 90's of the past century. The first step was Plotkin [22] recognizing that CPS translations can be used to simulate different evaluation orders within one another. In the 80's, Felleisen and his collaborators [5] made the connection between control operators in abstract machines and CPS translations, observing that the behavior of a control operator like call/cc in the *source language* of a CPS translation can be implemented by a purely functional expression in the *target language*. Griffin [13] observed the analogy of CPS translations and *negative translations* via the proofs-programs-correspondence, and through this analysis he discovered that the natural type for call/cc is *Pierce's law*,

[1] This work is supported by the Danish Council for Independent Research *Sapere Aude* grant "Complexity via Logic and Algebra" (COLA).

http://dx.doi.org/10.1016/j.entcs.2016.09.034

i.e. the propositional schema $((A \Rightarrow B) \Rightarrow A) \Rightarrow A$. Since Pierce's law when added to constructive logic yields full classical logic, his observation was celebrated as the unexpected discovery of an *algorithmic meaning of classical logic*.

Negative translations do not require full intuitionistic logic as target logic, and – inspired by Girard's [11] – Lafont, Reus, and Streicher identified the $(\neg, \wedge)$-fragment of intuitionistic logic as sufficient [18,19]. Although in this representation negation is taken as primitive, it is often useful to think of negation as given by the intuitionistic encoding $\neg A \equiv A \Rightarrow \bot$, and when constructing models in cartesian closed categories or *response categories* [24,23] $\mathbb{C}$, one has to interpret $\bot$ by an object $R \in \mathbb{C}$ other than the initial object to avoid degeneracy. This R is called the *response type*, and is comparable to the parameter A in Friedman's *A-translation* [9].

Krivine's *classical realizability* [17] is a realizability interpretation of classical logic which builds on the algorithmic understanding of classical logic arising from Griffin's insight. It is formulated using an extension of the λ-calculus with $\mathsf{call/cc}$, with an operational semantics provided by the *Krivine abstract machine* (KAM) [16]. To interpret logic, the interpretation utilizes a parameter called the *pole*, which plays a role comparable to the response type R, and to Friedman's A, as has been pointed out by Miquel [20].

A motivation of the present work is to make more explicit in which sense the pole plays the role of the response type, by giving a formulation of classical realizability in the *target language* instead of the source language, in which Krivine's work takes place. To this end, we introduce a term language for a minimal intuitionistic logic based on negation and disjunction (not *conjunction* as Lafont, Reus and Streicher proposed). A design goal is to get a *minimalistic system with a simple operational semantics*, and this is achieved by combining negation and disjunction into a 'synthetic' finitary multi-disjunction which should be understood as something like $\neg(A_1 \vee \cdots \vee A_n)$, but we write as $\langle \ell_1(A_1), \ldots, \ell_n(A_n) \rangle$, where $\ell_1, \ldots, \ell_n$ are elements of a countable set $\mathscr{L}$ of *labels*, comparable to *biases* in Girard's ludics [12]. The CPS target language is a term language of a natural deduction system based on this type constructor scheme.

Instead of presenting the system as a minimal intuitionistic logic based on negation and disjunction, we could also have chosen a presentation as a *dual-intuitionistic* (i.e. using sequents with many formulas on the right and at most one on the left) [27] system based on negation and *conjunction*, which would be closer to Carraro, Salibra, and Ehrhard's *stack calculus* [1], a system which was introduced for similar reasons (as an analysis of Krivine realizability), but is based on *implication* rather than negation. I have chosen the intuitionistic – rather than dual-intuitionistic – presentation for the simple reason that it is easier to handle and does not require as much 'backward thinking', but it is good to keep the alternative point of view in mind when comparing with Krivine realizability. In particular, the terms of the CPS target language are *records*, i.e. a kind of tuples, and should be viewed in analogy to *stacks* on the Krivine machine, which fits with the fact that we use sets of *terms* as truth values where Krivine uses sets of *stacks*.

However, we reverse the order on truth values relative to Krivine's account, and

take the empty set as falsity (rather than the set of all stacks as Krivine does), since we use a *call-by-value* translation of classical logic into the target language instead of the *call-by-name* translation that is implicit in Krivine's approach. This difference is immaterial from a model-theoretic point of view since it only reverses the order on predicates, which are symmetric as Boolean algebras, but it changes the implementation of classical connectives: where in Krivine realizability, *universal* quantification is the primitive operation that is given by unions of truth values (and the encoding of $\exists$ is indirect and involves dualization), in our presentation *existential quantification* is the primitive operation. Moreover, in Section 4 we describe how conjunction can be represented as an intersection type under certain (mild) conditions, and together we get a simple representation of the connectives of regular logic (i.e. the $(\exists, \wedge, \top)$-fragment of first order logic) not involving the pole at all. This is desirable since regular logic is all that is required for the *tripos-to-topos construction* [14], and a simpler representation of its connectives greatly facilitates calculations in classical realizability toposes.

1.1 Related work

The CPS target language is similar in spirit to Thielecke's *CPS calculus* [26], which can also be motivated as a term calculus for a type system with a kind of multi-negation. The main difference is that in Thielecke's system the basic type constructor is a negated n-ary *conjunction*, and not a an n-ary disjunction as in the CPS target language.

Although different in objective, Curien et al.'s work on term calculi for classical logic [2,3] was inspirational for the present article, and so were Melliès and Tabareau's *tensor logic* [21] and Zeilberger's analysis of polarized logic [28].

Terui's *computational ludics* [25] is a term calculus for ludics designs with a notion of head reduction analogous to the CPS target language. Specifically, the CPS target language can be understood as a non-linear version of the purely additive fragment of the syntax of computational ludics.

Finally – and rather unsurprisingly – there is a strong analogy to Hyland and Ong's *innocent strategies* [15]. Specifically, the η-expanded closed normal forms of a type A without variables are precisely the innocent strategies on A viewed as tree.

2 The CPS target language

The syntax of the CPS target language, given in Table 1, distinguishes two syntactic classes called *terms* and *programs*.

A *term* is either a variable or a *record*, i.e. a family $\langle \ell_1(x_1.\, p_1), \ldots, \ell_n(x_n.\, p_n)\rangle$ of programs p_i – the *methods* of the record, each abstracted by a variable x_i – indexed by a finite subset $\{\ell_1, \ldots, \ell_n\} \subseteq \mathscr{L}$ of a countable set of *labels*, which we take to be the set $\mathscr{L} = \{\mathsf{a}, \ldots, \mathsf{z}\}^*$ of lower case strings (in practice we will only use strings of length 1). The use of different fonts is important: curly ℓ, ℓ are placeholders for generic labels, whereas sans-serif k, l are specific labels. The order in which the methods of a record are listed is not important – we view them abstractly as

Expressions:

$$\textit{Terms:} \quad s,t,u ::= \quad x \mid \langle\, \ell_1(x.\,p_1), \ldots,\, \ell_n(x.\,p_n)\rangle$$

$$\textit{Programs:} \quad p,q ::= \quad t_\ell u \mid \ldots \textit{ (possibly non-logical instructions)}$$

Reduction:

$$\langle\, \ell_1(x.\,p_1), \ldots,\, \ell_n(x.\,p_n)\rangle_{\ell_i} t \quad \succ \quad p_i[t/x] \qquad \text{if } 1 \le i \le n$$

Types:

$$A \quad ::= \quad X \mid \big\langle \ell_1(A_1), \ldots, \ell_n(A_n) \big\rangle \qquad n \ge 0$$

Typing rules:

$$\text{(Var)} \quad \frac{}{\Gamma \;\vdash\; x_i : A_i} \qquad \Gamma \equiv x_1 : A_1, \ldots, x_n : A_n,\;\; 1 \le i \le n$$

$$\text{(Abs)} \quad \frac{\Gamma, y : B_1 \;\vdash\; p_1 \qquad \cdots \qquad \Gamma, y : B_m \;\vdash\; p_m}{\Gamma \;\vdash\; \langle\, \ell_1(y.\,p_1), \ldots,\, \ell_m(y.\,p_m)\rangle : \big\langle \ell_1(B_1), \ldots, \ell_m(B_m)\big\rangle}$$

$$\text{(App)} \quad \frac{\Gamma \;\vdash\; t : \big\langle \ell_1(B_1), \ldots, \ell_m(B_m)\big\rangle \qquad \Gamma \;\vdash\; u : B_i}{\Gamma \;\vdash\; t_{\ell_i} u} \quad 1 \le i \le m$$

Table 1
The CPS target language.

functions from finite sets $F \subseteq_{\text{fin}} \mathscr{L}$ of labels to programs with a distinguished free variable. In accordance with this viewpoint, we use 'family notation' $\langle\, \ell(x.\,p) \mid \ell \in F\rangle$ for records when convenient (in particular in Section 4). We refer to the set of labels indexing the methods of a record t as the *domain* of the record and denote it $\mathrm{dom}(t)$ – thus $\mathrm{dom}(\langle\, \ell_1(x_1.\,p_1), \ldots,\, \ell_n(x_n.\,p_n)\rangle) = \{\ell_1, \ldots, \ell_n\}$ and $\mathrm{dom}(\langle\, \ell(x.\,p) \mid \ell \in F\rangle) = F$.

A *program* is an expression of the form $t_\ell u$, with the intended meaning that the program (or *method*) labeled ℓ in t is called with u as an argument. This reading suggests the reduction rule $\langle\, \ell_1(x_1.\,p_1), \ldots,\, \ell_n(x_n.\,p_n)\rangle_{\ell_i} t \succ p_i[t/x_i]$ (provided $1 \le i \le n$), which gives the operational semantics of the language. We use the symbol '$\succ$' only for top-level reduction of programs (i.e. *weak head reduction*), and write '$\to_\beta$' for the compatible closure (i.e. the closure under term and program formers) of $\succ$ on terms and programs. A *redex* is a program $t_\ell u$ where t is a record (not a variable). A redex $t_\ell u$ with $\ell \notin \mathrm{dom}(t)$ can not be reduced and is said to be *blocked*. A *normal form* is a term or program that does not contain any redexes, i.e. in every application $t_\ell u$ the term t is a variable.

We define the sets $\mathrm{FV}(t)$ and $\mathrm{FV}(p)$ of *free variables* of a term or program in the usual way, where the distinguished variable x of a method $\ell(x.\,p)$ in a record t is considered bound in p. There are no closed normal programs (since the term in head position cannot be a variable) but there are blocked closed programs like $\langle\rangle_{\mathsf{k}}\langle\rangle$

and diverging closed programs like $\langle\, \mathsf{k}(x.\, x_{\mathsf{k}} x)\rangle_{\mathsf{k}} \langle\, \mathsf{k}(x.\, x_{\mathsf{k}} x)\rangle$.

To allow the construction of non-trivial classical realizability models, the syntax has to be extended by non-logical constructs like *constants* or *instructions* to perform side effects [2]. This is achieved by extending the clause for programs in the grammar. To have a model for idealized shell-programs, for example, one can extend the definition of programs to be

$$p, q \quad ::= \quad t_\ell u \mid \mathbf{r}(p, q) \mid \mathbf{w0}(p) \mid \mathbf{w1}(p) \mid \mathbf{0} \mid \mathbf{1}$$

with the intended meaning that the program $\mathbf{r}(p, q)$ reads a bit from standard input and continues with p or q depending on its value, $\mathbf{w0}(p)$ and $\mathbf{w1}(p)$ write a 0 or 1, respectively, to standard output before continuing with p, and $\mathbf{0}$ and $\mathbf{1}$ represent successful and unsuccessful termination. For example, $\langle\, \mathsf{k}(x.\, x_{\mathsf{k}} x)\rangle_{\mathsf{k}} \langle\, \mathsf{k}(x.\, \mathbf{r}(x_{\mathsf{k}} x, \mathbf{0}))\rangle$ is a program that reads bits from standard input until it encounters a 1, whereupon it terminates successfully.

Formally, such an extension of the syntax has to be accompanied by an extension of the operational semantics, which in the case of the above example can either be given as a labeled transition system or as a transition relation on programs with *state*. This is explained in detail in [8] using Krivine's syntax, where it is also explained how in such a setting specifications on program behavior give rise to poles and thus to realizability triposes and toposes. These ideas all transfer to the reformulation of classical realizability given in this article, but instead of formulating our results in this generality – which would require a lot of repetition – we use as running example only a single non-logical constant **end** which represents termination and is comparable to Girard's *daimon* ✠ [3]. Thus, from now on we assume that programs are of the form

$$p, q \quad ::= \quad t_\ell u \mid \mathbf{end}.$$

We denote the sets of closed terms and programs generated by this grammar (together with the rule for terms in Table 1) by $\mathbb{T}$ and $\mathbb{P}$, and more generally we denote by $\mathbb{T}[x_1, \ldots, x_n]$ and $\mathbb{P}[x_1, \ldots, x_n]$ the sets of terms and programs whose free variables are contained in $\{x_1, \ldots, x_n\}$. The analogous sets of *pure* terms and programs (i.e. those not containing **end**) are denoted by $\mathbb{T}_0$, $\mathbb{P}_0$, $\mathbb{T}_0[x_1, \ldots, x_n]$, and $\mathbb{P}_0[x_1, \ldots, x_n]$.

We consider a Curry-style type system for the CPS target language, whose types are generated from type variables and for each finite set $\{\ell_1, \ldots, \ell_n\}$ an n-ary constructor which forms the record type $\langle \ell_1(A_1), \ldots, \ell_n(A_n)\rangle$ out of types $A_1, \ldots, A_n$. There are two kinds of typing judgments corresponding to the two syntactic classes:

- *terms* $t \in \mathbb{T}_0[x_1, \ldots, x_n]$ are typed by sequents $(\, x_1 : A_1, \ldots, x_n : A_n \vdash t : B\,)$, and
- *programs* $p \in \mathbb{P}_0[x_1, \ldots, x_n]$ are typed by sequents $(\, x_1 : A_1, \ldots, x_n : A_n \vdash p\,)$.

[2] Essentially because of [8, Lemma 26].

[3] A referee points out that a concept comparable to the daimon already appears in Coquand's *evidence semantics* [4].

(Cut)	$\dfrac{\Gamma \vdash s : A \qquad \Gamma, x : A \vdash p}{\Gamma \vdash p[s/x]}$	$\dfrac{\Gamma \vdash s : A \qquad \Gamma, x : A \vdash t : B}{\Gamma \vdash t[s/x] : B}$
(Sym)	$\dfrac{\Gamma \vdash p}{\sigma(\Gamma) \vdash p}$	$\dfrac{\Gamma \vdash t : B}{\sigma(\Gamma) \vdash t : B}$
(Weak)	$\dfrac{\Gamma \vdash p}{\Gamma, x : A \vdash p}$	$\dfrac{\Gamma \vdash t : B}{\Gamma, x : A \vdash t : B}$
(Contr)	$\dfrac{\Gamma, x : A, y : A \vdash p}{\Gamma, x : A \vdash p[x/y]}$	$\dfrac{\Gamma, x : A, y : A \vdash t : B}{\Gamma, x : A \vdash t[x/y] : B}$

Table 2
Admissible rules for the typing relation, where $\Gamma \equiv x_1 : A_1, \dots, x_n : A_n$, and σ is a permutation.

Thus, programs are not associated to types, but we think of them as having response type (or type $\bot$).

There are three rules (Var), (Abs) and (App), typing *variables*, *records*, and *applications*, respectively. Furthermore, the typing relation is closed under a number of *admissible rules*.

Lemma 2.1 *The derivable typing judgments are closed under the rules in Table 2.*

Proof. Each of the four pairs of rules can be shown to be admissible by simultaneous induction on the structure of t and p. □

A consequence of the admissibility of (Cut) is *subject reduction*.

Lemma 2.2 (Subject reduction) *If* $\Gamma \vdash \langle \ell_1(x_1.\, p_1), \dots, \ell_n(x_n.\, p_n)\rangle_{\ell_i} t$ *is derivable for some* $1 \leq i \leq n$, *then* $\Gamma \vdash p_i[t/x_i]$ *is derivable.*

Proof. Inspection of the typing rules shows that $\Gamma \vdash \langle \ell_1(x_1.\, p_1), \dots, \ell_n(x_n.\, p_n)\rangle_{\ell_i} t$ can only be derived by a deduction

$$\dfrac{\dfrac{\Gamma, x_1 : A_1 \vdash p_1 \qquad \dots \qquad \Gamma, x_n : A_n \vdash p_n}{\Gamma \vdash \langle \ell_1(x_1.\, p_1), \dots, \ell_n(x_n.\, p_n)\rangle : \langle \ell_1(A_1), \dots, \ell_n(A_n)\rangle} \qquad \Gamma \vdash t : A_i}{\Gamma \vdash \langle \ell_1(x_1.\, p_1), \dots, \ell_n(x_n.\, p_n)\rangle_{\ell_i} t}$$

and applying (Cut) to the hypotheses with p_i and t yields the claim. □

3 Realizability

Classical realizability models are always defined relative to a *pole*, which is a set $\perp\!\!\!\perp \subseteq \mathbb{P}$ of closed programs satisfying

$$p \succ q, q \in \perp\!\!\!\perp \;\Rightarrow\; p \in \perp\!\!\!\perp \tag{1}$$

for all $p, q \in \mathbb{P}$. The deliberations that follow are valid for arbitrary poles satisfying this condition (relative to reasonable extensions of the pure language with non-logical instructions such as in [8,10]), but to have something to hold on to, we fix a

pole $\perp\!\!\!\perp$ by

$$\perp\!\!\!\perp \;=\; \{p \mid p \succ^* \mathbf{end}\}\,,$$

which is the set of all programs p whose weak reduction sequence 'terminates', i.e. leads to the constant **end** [4].

A *truth value* is a set $S \subseteq \mathbb{T}$ of closed terms. We define as semantic counterparts of the type constructors for each set $\{\ell_1, \ldots, \ell_n\}$ of labels an n-ary connective on the set $P(\mathbb{T})$ of truth values.

Definition 3.1 Given truth values $S_1, \ldots, S_n \in P(\mathbb{T})$ and labels $\ell_1, \ldots, \ell_n \in \mathscr{L}$, the truth value $\langle \ell_1(S_1), \ldots, \ell_n(S_n) \rangle$ is defined by

$$\langle \ell_1(S_1), \ldots, \ell_n(S_n) \rangle = \{t \in \mathbb{T} \mid \forall i \in \{1, \ldots, n\}\; \forall s \in S_i \,.\, t_{\ell_i} s \in \perp\!\!\!\perp\}\,.$$

We introduce *realization judgments* as semantic counterparts of typing judgments.

Definition 3.2 Given truth values $S_1, \ldots, S_n, T \subseteq \mathbb{T}$, and a term $t \in \mathbb{T}[x_1, \ldots, x_n]$ or program $p \in \mathbb{P}[x_1, \ldots, x_n]$,

$$\begin{aligned} \text{the notation} \quad & x_1 : S_1, \ldots, x_n : S_n \Vdash t : T && (2)\\ \text{stands for} \quad & \forall s_1 \in S_1, \ldots, s_n \in S_n \,.\, t[s_1/x_1, \ldots, s_n/x_n] \in T \\ \text{and the notation} \quad & x_1 : S_1, \ldots, x_n : S_n \Vdash p && (3)\\ \text{stands for} \quad & \forall s_1 \in S_1, \ldots, s_n \in S_n \,.\, p[s_1/x_1, \ldots, s_n/x_n] \in \perp\!\!\!\perp. \end{aligned}$$

We call expressions of the form (2) and (3) *realization judgments*. Slightly redundantly, we also say '*the realization judgment* ($\Gamma \Vdash t : T$) *is valid*' instead of simply asserting the judgment itself.

The following result is an analogue of Krivine's *adequation lemma* [17, Theorem 3].

Lemma 3.3 *Valid realization judgments are closed under the rules in Table 3.*

Proof. The only nontrivial case is (Abs). Assume that $\Gamma, y : T_k \Vdash p_k$ for $1 \le k \le m$, and that $s_i \in S_i$ for $1 \le i \le n$. We have to show that

$$(\langle \ell_1(y.\, p_1), \ldots,\, \ell_n(y.\, p_m) \rangle [\vec{s}/\vec{x}])_{\ell_j} t \in \perp\!\!\!\perp$$

for every $1 \le j \le m$ and $t \in T_j$. For fixed j and t we have

$$\begin{aligned} (\langle \ell_1(y.\, p_1), \ldots,\, \ell_n(y.\, p_m) \rangle [\vec{s}/\vec{x}])_{\ell_j} t = \\ (\langle \ell_1(y.\, p_1[\vec{s}/\vec{x}]), \ldots,\, \ell_n(y.\, p_m[\vec{s}/\vec{x}]) \rangle)_{\ell_j} t \succ p_j[\vec{s}/\vec{x}, t/y] \end{aligned}$$

where the reduct is in $\perp\!\!\!\perp$ by assumption, and the claim follows from (1). □

[4] The classical realizability model arising from this pole has some interesting properties, as the author learned from Krivine [7].

$$\text{(Var)}\quad \frac{}{\Gamma \Vdash x_i : S_i}$$

$$\text{(App)}\quad \frac{\Gamma \Vdash t : \langle \ell_1(T_1), \ldots, \ell_n(T_n)\rangle \qquad \Gamma \Vdash u : T_i}{\Gamma \Vdash t_{\ell_i} u}$$

$$\text{(Abs)}\quad \frac{\Gamma, y : T_1 \Vdash p_1 \qquad \cdots \qquad \Gamma, y : T_m \Vdash p_m}{\Gamma \Vdash \langle \ell_1(y.\, p_1), \ldots, \ell_n(y.\, p_m)\rangle : \langle \ell_1(T_1), \ldots, \ell_n(T_m)\rangle}$$

$$\text{(Cut)}\quad \frac{\Gamma \Vdash s : S \qquad \Gamma, x : S \Vdash p}{\Gamma \Vdash p[s/x]} \qquad \frac{\Gamma \Vdash s : S \qquad \Gamma, x : S \Vdash t : T}{\Gamma \Vdash t[s/x] : T}$$

$$\text{(Sym)}\quad \frac{\Gamma \Vdash p}{\sigma(\Gamma) \Vdash p} \qquad \frac{\Gamma \Vdash t : T}{\sigma(\Gamma) \Vdash t : T}$$

$$\text{(Weak)}\quad \frac{\Gamma \Vdash p}{\Gamma, x : S \Vdash p} \qquad \frac{\Gamma \Vdash t : T}{\Gamma, x : S \Vdash t : T}$$

$$\text{(Contr)}\quad \frac{\Gamma, x : S, y : S \Vdash p}{\Gamma, x : S \Vdash p[x/y]} \qquad \frac{\Gamma, x : S, y : S \Vdash t : T}{\Gamma, x : S \Vdash t[x/y] : T}$$

Table 3
Admissible rules for realization judgments, where $S_1, \ldots, S_n, S, T_1, \ldots, T_m, T \subseteq \mathbb{T}$, $\Gamma \equiv x_1 : S_1, \ldots, x_n : S_n$, and σ is a permutation.

3.1 Classical realizability triposes

We now show how to do classical realizability in the CPS target language by instantiating a simple (call-by-value) negative translation. To start we fix the shorthands

$$\top \equiv \langle\rangle \qquad \neg A \equiv \langle \mathsf{k}(A)\rangle \qquad \neg(A, B) \equiv \langle \mathsf{l}(A), \mathsf{r}(B)\rangle$$

for nullary, unary, and binary type constructors, and using these we encode classical conjunction as

$$A \wedge B \equiv \neg(\neg A, \neg B). \tag{4}$$

The negative translation maps *classical sequents*

$$A_1, \ldots, A_n \vdash B_1, \ldots, B_m$$

consisting of formulas built up from propositional variables and the connectives, $\top$, $\neg$ and $\wedge$, to *intuitionistic sequents*

$$A_1^*, \ldots, A_n^*, \neg B_1^*, \ldots, \neg B_m^* \vdash$$

where the formulas A_i^* and B_j^* are obtained by expanding the classical connectives according to the above shorthands and encoding.

We could now define classical realization judgments by mimicking the negative translation on the level of realizability, but we will not spell this out explicitly, and

rather develop the remainder of the section in categorical language, by laying out the construction of *classical realizability triposes* analogous to the treatment in [8].

Broadly speaking, *realizability triposes* [14] capture the model theoretic essence of realizability interpretations as a collection of order relations on sets of *semantic predicates*, which together are required to form an *indexed preorder* – i.e. a contravariant functor $\mathscr{P} : \mathbf{Set}^{\mathsf{op}} \to \mathbf{Ord}$ from sets to preorders – subject to certain conditions. The precise definition of *strict Boolean tripos* (which is the version of triposes that we use) is given in Definition A.3.

In our setting, *semantic predicates* on a set J are functions

$$\varphi, \psi : J \to P(\mathbb{T})$$

into the set of truth values, and the order on predicates is defined by

$$\varphi \leq \psi \quad :\Leftrightarrow \quad \exists p \in \mathbb{P}_0[x,y]\ \forall j \in J\ .\ \big(\, x : \varphi(j), y : \neg\psi(j) \Vdash p \,\big), \tag{5}$$

i.e. $\varphi \leq \psi$ if there exists a *pure* program $p[x,y]$ which realizes the negative translation of $\varphi(j) \vdash \psi(j)$ uniformly in j.

The first step in establishing that semantic predicates form a tripos is to show that the predicates on a fixed set form a *Boolean prealgebra*, i.e. a preorder whose poset reflection is a Boolean algebra (Definition A.1).

Theorem 3.4 *For every set J, the set of $P(\mathbb{T})^J$ of semantic predicates on J equipped with the order relation (5) is a Boolean prealgebra.*

Proof. We show first that $\leq$ is actually a preorder. Reflexivity follows from the fact that $(\, x : S, y : \neg S \Vdash y\mathsf{k}x \,)$ for arbitrary truth values S.

For transitivity, assume that $\varphi \leq \psi$ and $\psi \leq \theta$, i.e. that there exist $p \in \mathbb{P}[v,w]$ and $q \in \mathbb{P}[x,y]$ such that $(\, v : \varphi(j), w : \neg\psi(j) \Vdash p \,)$ and $(\, x : \psi(j), y : \neg\theta(j) \Vdash q \,)$.

The claim $\varphi \leq \theta$ follows from Lemma 3.3 via the derivation

$$\dfrac{\dfrac{x : \psi(j), y : \neg\theta(j) \Vdash q}{y : \neg\theta(j) \Vdash \langle\, \mathsf{k}(x.\, q) \rangle : \neg\psi(j)} \qquad v : \varphi(j), w : \neg\psi(j) \Vdash p}{v : \varphi(j), y : \neg\theta(j) \Vdash p[\langle\, \mathsf{k}(x.\, q) \rangle / w]}$$

Next we show that the order has finite meets. The predicate with value constant $\top$ is a greatest element, since $(\, x : S, y : \neg\top \Vdash y_{\mathsf{k}} \langle\, \rangle)$ for arbitrary truth values S. We claim that a binary meet of φ and ψ is given by pointwise application of (the semantic version of) the type constructor defined in (4), i.e. $(\varphi \wedge \psi)(j) = \varphi(i) \wedge \psi(i)$. The such defined $\varphi \wedge \psi$ is smaller than φ since $(\, x : \neg(\neg\varphi(j), \neg\psi(j)), y : \neg\varphi(j) \Vdash x_{\mathsf{l}} y \,)$, and similarly for ψ. To see that it is a *greatest* lower bound, assume that $\theta \leq \varphi$ and $\theta \leq \psi$, i.e. there exist programs $p \in \mathbb{P}[w,x]$ and $q \in \mathbb{P}[w,y]$ such that $(\, w : \theta(j), x : \neg\varphi(j) \Vdash p \,)$ and $(\, w : \theta(j), y : \neg\psi(j) \Vdash q \,)$. Then we have $\theta \leq \varphi \wedge \psi$ by

the following derivation.

$$\dfrac{\dfrac{w:\theta(j), x:\neg\varphi(j) \Vdash p \qquad w:\theta(j), y:\neg\psi(j) \Vdash q}{w:\theta(j) \Vdash \langle\, \mathsf{l}(x.\, p),\, \mathsf{r}(y.\, q)\rangle : \neg(\neg\varphi(j), \neg\psi(j))}}{w:\theta(j), z:\neg(\neg\varphi(j), \neg\psi(j)) \Vdash z_{\mathsf{k}}\langle\, \mathsf{l}(x.\, p),\, \mathsf{r}(y.\, q)\rangle}$$

To finish the proof that $(P(\mathbb{P})^J, \leq)$ is a Boolean algebra, it now suffices to verify the conditions (i)–(iii) of Lemma A.2, with the negation operation given by $(\neg\varphi)(j) = \neg\varphi(j)$.

For (i) assume that $\varphi \wedge \psi \leq \neg\bot$, i.e. that there exists $p[x, y] \in \mathbb{P}[x, y]$ with $(\, x:\neg(\neg\varphi(j), \neg\psi(j)), y:\neg\neg\top \Vdash p\,)$. Then we have

$$w:\varphi(j), z:\neg\neg\psi(j) \Vdash z_{\mathsf{k}}\langle\, \mathsf{k}(y.\, p[\langle\, \mathsf{l}(v.\, v_{\mathsf{k}} w),\, \mathsf{r}(w.\, w_{\mathsf{k}} y)\rangle/x, \langle\, \mathsf{k}(v.\, v_{\mathsf{k}}\langle\rangle)\rangle/y])\rangle$$

(in the following we do not spell out the derivation of realization judgments any more, and leave the type checking to the reader) and hence $\varphi \leq \neg\psi$.

For (ii) we have

$$x:\neg(\neg\varphi(j), \neg\neg(\varphi(j)), y:\neg\neg\top \Vdash x_{\mathsf{r}}\langle\, \mathsf{k}(z.\, x_{\mathsf{l}} z)\rangle$$

and for (iii) we have

$$x:\neg\neg\varphi(j), y:\neg\varphi(j) \Vdash x_{\mathsf{k}} y.$$

□

Every function $f : J \to I$ induces a function $f^* : P(\mathbb{T})^I \to P(\mathbb{T})^J$ on predicates by precomposition, and it is easy to see that f^* is monotone and preserves all logical structure (since all propositional operations on predicates are defined pointwise in a uniform way). Since the operation $(f \mapsto f^*)$ clearly preserves composition and identities, it is the morphism part of a contravariant functor

$$\mathscr{K}_{\perp\!\!\!\perp} : \mathbf{Set}^{\mathsf{op}} \to \mathbf{BA}.$$

from sets to Boolean prealgebras with object part $J \mapsto (P(\mathbb{T})^J, \leq)$. We can now prove the main theorem.

Theorem 3.5 *$\mathscr{K}_{\perp\!\!\!\perp}$ is a strict Boolean tripos (Definition A.3).*

Proof. It remains to show that the reindexing maps f^* admit left adjoints subject to the Beck-Chevalley condition, and that there is a generic predicate.

Let $f : J \to I$. We claim that a left adjoint $\exists_f$ to f^* can be defined by fiberwise union, i.e.

$$\exists_f(\varphi)(i) = \bigcup_{fj=i} \varphi(j) \quad \text{for} \quad \varphi \in P(\mathbb{T})^J,$$

and to prove this we have to show that for any $\psi \in P(\mathbb{T})^I$ we have $\varphi \leq f^*\psi$ if and only if $\exists_f\varphi \leq \psi$. Unfolding definitions yields

$$\exists p \in \mathbb{P}_0[x, y]\ \forall j \in J\ \forall s \in \varphi(j)\ \forall t \in \neg\psi(fj)\,.\, p[s, t] \in \perp\!\!\!\perp$$

for the first inequality, and

$$\exists p \in \mathbb{P}_0[x,y]\ \forall i \in I\ \forall s \in \bigcup_{fj=i} \varphi(j)\ \forall t \in \neg\psi(i)\,.\,p[s,t] \in \perp\!\!\!\perp$$

for the second one. The two statements are equivalent since in both cases the arguments of φ and ψ range over all pairs (i,j) with $fj = i$.

It is easy to see (and well known e.g. from the effective tripos) that fiberwise unions strictly satisfy the Beck-Chevalley condition.

Finally, a generic predicate is given by the identity function on $P(\mathbb{T})$. □

To conclude the section, we reprove [8, Lemma 26] in the new syntax.

Lemma 3.6 *The tripos $\mathcal{K}_{\perp\!\!\!\perp}$ induced by a pole $\perp\!\!\!\perp$ is non-degenerate (not equivalent to the terminal tripos) if and only if $\mathbb{P}_0 \cap \perp\!\!\!\perp = \varnothing$.*

Proof. A tripos is degenerate if and only if all truth values are equivalent, which is easily seen to be equivalent to the existence of a pure program $p[x] \in \mathbb{P}_0[x]$ such that the realization judgment $(\, x : \mathbb{T} \Vdash p[x] \,)$ holds. If this is the case, then $p[\langle\rangle] \in \mathbb{P}_0 \cap \perp\!\!\!\perp$. Conversely, if there exists $q \in \mathbb{P}_0 \cap \perp\!\!\!\perp$ then we have $(\, x : \mathbb{T} \Vdash q \,)$. □

4 Conjunction as intersection type

In the previous section we have seen that relative to a fixed pole $\perp\!\!\!\perp$ the semantic predicates give rise to a tripos $\mathcal{K}_{\perp\!\!\!\perp}$, and this tripos in turn gives rise to a *topos* $\mathbf{Set}[\perp\!\!\!\perp]$ whose construction relies only on the *regular* fragment of first order logic, i.e. the fragment of logic consisting of existential quantification and conjunction. To facilitate computation in classical realizability toposes, it is good to have an easy representations of the basic connectives, and in the proof of Theorem 3.5 we saw that existential quantification in the tripos is given by set theoretic union, which is easy enough. However, for conjunction we only have the representation (4) and the involved double negation entails a high logical complexity and obscures things considerably, i.e. it is difficult to know what the elements of $S \wedge T$ look like, even if we know the elements of S and T very well.

In this section, we show that under certain conditions on the pole we can identify a class of 'nice' representatives of predicates in the tripos which admits an implementation of conjunction as intersection type, while being closed under the other logical operations. The idea to represent conjunction as intersection is inspired by ludics [12].

Given a record

$$t = \langle\, \ell(x.p) \mid \ell \in F \rangle$$

and a set $M \subseteq \mathscr{L}$ of labels, define the *restriction of t to M* to be the record

$$t|_M \;=\; \langle\, \ell(x.p) \mid \ell \in F \cap M \rangle.$$

The *syntactic order* $\sqsubseteq$ on terms and programs is the reflexive-transitive and compatible (i.e. closed under term and program constructors) closure of the set of all

pairs $(t|_M, t)$ for records t and sets M of labels. Observe that the empty record $\langle\rangle$ is smaller than any other record in the syntactic order, but not smaller than a variable.

Definition 4.1 A *pole* $\perp\!\!\!\perp$ is called *strongly closed*, if it satisfies the conditions

$$p \to_\beta^* q, q \in \perp\!\!\!\perp \;\Rightarrow\; p \in \perp\!\!\!\perp \quad \text{and}$$
$$p \sqsubseteq q, p \in \perp\!\!\!\perp \;\Rightarrow\; q \in \perp\!\!\!\perp,$$

i.e. it is closed under inverse β-reduction and upward w.r.t. the syntactic order.

A *truth value* $S \subseteq \mathbb{T}$ is called *strongly closed*, if it satisfies the analogous conditions

$$t \to_\beta^* u, u \in S \;\Rightarrow\; t \in S \quad \text{and}$$
$$t \sqsubseteq u, t \in S \;\Rightarrow\; u \in S.$$

Although strong closure is a much stronger condition on a pole than mere closure under inverse head reduction, it is satisfied for many 'reasonable' poles, in particular for the pole of terminating programs, and more generally for poles constructed from specifications as in [8].

For a fixed strongly closed $\perp\!\!\!\perp$, there is an easy way to strongly close any given truth value, via a well-known double duality construction. Concretely, for $S \subseteq \mathbb{T}$ define

$$S^\uparrow = \{p[x] \in \mathbb{P}[x] \mid \forall s \in S \,.\, p[s] \in \perp\!\!\!\perp\}\,,$$

and dually for $E \subseteq \mathbb{P}[x]$ define

$$S^\downarrow = \{s \in \mathbb{T} \mid \forall p[x] \in E \,.\, p[s] \in \perp\!\!\!\perp\}\,.$$

If $\perp\!\!\!\perp$ is strongly closed, it is obvious that so is $S^{\uparrow\downarrow}$ for any truth value S.

A truth value S is said to be *supported* by a set $M \subseteq \mathscr{L}$ of labels, if we have $s|_M \in S$ for every $s \in S$. More generally, a predicate $\varphi \in P(\mathbb{T})^J$ is said to be supported by M, if $\varphi(j)$ is supported by M for all $j \in J$.

The main result of the section is the following.

Theorem 4.2 *Let $\varphi, \psi \in P(\mathbb{T})^J$ be predicates that are both pointwise strongly closed, and supported by disjoint finite sets $F = \{\ell_1, \ldots, \ell_n\}$ and $G = \{k_1, \ldots, k_m\}$ of labels, respectively. Then the predicate $\varphi \cap \psi$, which is defined by $(\varphi \cap \psi)(j) = \varphi(j) \cap \psi(j)$, is a meet of φ and ψ and is supported by $F \cup G$.*

Proof. We claim that the realization judgments

$$x : \varphi(j) \cap \psi(j) \Vdash x : \varphi(j) \qquad\qquad x : \varphi(j) \cap \psi(j) \Vdash x : \psi(j)$$

and

$$x : \varphi(j), y : \psi(j) \Vdash u[x, y] : \varphi(j) \cap \psi(j)$$
$$\text{with} \quad u[x, y] = \langle\, \ell_1(z.\, x_{\ell_1} z), \ldots,\, \ell_n(z.\, x_{\ell_n} z),\, k_1(z.\, y_{k_1} z), \ldots,\, k_m(z.\, y_{k_m} z)\rangle$$

hold for all j. The first two are obvious. For the third one assume that $s \in \varphi(j)$ and $t \in \psi(j)$. Then for each $\ell_i \in \mathrm{dom}(s)$ the redex $s_{\ell_i} z$ in $u[s,t]$ can be reduced, and the result $u'[s,t]$ satisfies $u'[s,t] \sqsupseteq s|_F$. We have $s|_F \in \varphi(j)$ since $\varphi(j)$ is supported by F, and $u'[s,t] \in \varphi(j) \in \varphi(j)$ and $u[s,t] \in \varphi(j)$ by strong closure. An analogous argument shows that $u[s,t]$ is in $\psi(j)$, and therefore in $\varphi(j) \cap \psi(j)$. The claim that $\varphi \cap \psi$ is a meet of φ and ψ now follows from the next lemma.

To see that $\varphi \cap \psi$ is supported by $F \cup G$, assume that $t \in \varphi(j) \cap \psi(j)$ for some $j \in J$. Then $t|_{F\cup G} \sqsupseteq t|_F \in \varphi(j)$ and by strong closure we have $t|_{F\cup G} \in \varphi(j)$. □

Lemma 4.3 *If $\varphi, \psi, \theta \in P(\mathbb{T})^J$ are predicates and $s[z], t[z] \in \mathbb{T}_0[z]$ and $u[x,y] \in \mathbb{T}_0[x,y]$ are pure terms such that the realization judgments*

$$z : \theta(j) \Vdash s[z] : \varphi(x) \qquad z : \theta(j) \Vdash t[z] : \psi(x) \qquad x : \varphi(j), y : \psi(j) \Vdash u[x,y] : \theta(x,y)$$

for all $j \in J$, then θ is a meet of φ and ψ.

Proof. From the first two judgments we can deduce $(\, z : \theta(j), v : \neg\varphi(j) \Vdash v_{\mathsf{k}} s[z] \,)$ and $(\, z : \theta(j), v : \neg\psi(j) \Vdash v_{\mathsf{k}} t[z] \,)$, which means that $\theta \leq \varphi$ and $\theta \leq \psi$, and thus $\theta \leq \varphi \wedge \psi$. From the third judgment we can derive

$$w : \neg(\neg\varphi(j), \neg\psi(j)), z : \neg\theta(j) \Vdash w_{\mathsf{l}} \langle\, \mathsf{k}(x.\, w_{\mathsf{r}} \langle\, \mathsf{k}(y.\, z_{\mathsf{k}} u[x,y]) \rangle) \rangle,$$

which means that $\varphi \wedge \psi \leq \theta$. □

Thus we have a nice representation of conjunction for pointwise strongly closed predicates which are finitely supported by disjoint sets.

Disjointness can always be achieved by renaming, i.e. 'relocating', as long as supports are *finite*. Moreover, strong closure and finite support are preserved by existential quantification, and by the semantic type constructors (Definition 3.1) provided the the pole is strongly closed. A finitely supported and strongly closed generic predicate can also be obtained, by negating the canonical one given by the identity on $P(\mathbb{T})$.

Acknowledgement

The ideas presented in this article were developed over a long period of time, and I profited from discussions on related issues with many people, including – but not limited to – Pierre Clairambault, Pierre-Louis Curien, Nicolas Guenot, Paul Blain Levy, Paul-André Melliès, Guillaume Munch-Maccagnoni, Jakob Grue Simonsen, Thomas Streicher, Noam Zeilberger, and Stéphane "El Zím" Zimmermann.

Thanks to the referees for their careful rereading and helpful comments.

References

[1] A. Carraro, T. Ehrhard, and A. Salibra. The stack calculus. In *Proceedings Seventh Workshop on Logical and Semantic Frameworks, with Applications, LSFA 2012, Rio de Janeiro, Brazil, September 29-30, 2012.*, pages 93–108, 2012.

[2] P.L. Curien and H. Herbelin. The duality of computation. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), Montreal, Canada, September 18-21, 2000.*, pages 233–243, 2000.

[3] P.L. Curien and G. Munch-Maccagnoni. The duality of computation under focus. In *Theoretical computer science*, volume 323 of *IFIP Adv. Inf. Commun. Technol.*, pages 165–181. Springer, Berlin, 2010.

[4] T. Coquand. A semantics of evidence for classical arithmetic. *J. Symbolic Logic*, 60(1):325–337, 1995.

[5] M. Felleisen, D. Friedman, E. Kohlbecker, and B. Duba. Reasoning with continuations. In *Proceedings of the Symposium on Logic in Computer Science (LICS '86), Cambridge, Massachusetts, USA, June 16-18, 1986*, pages 131–141, 1986.

[6] R.C. Flagg. Church's thesis is consistent with epistemic arithmetic. In *Intensional mathematics*, volume 113 of *Stud. Logic Found. Math.*, pages 121–172. North-Holland, Amsterdam, 1985.

[7] J. Frey. Computability and Krivine realizability. Note of a conversation with J.L. Krivine, available at https://sites.google.com/site/jonasfreysite/krivine-comp.pdf, 2015.

[8] J. Frey. Realizability toposes from specifications. In *13th International Conference on Typed Lambda Calculi and Applications, TLCA 2015, July 1-3, 2015, Warsaw, Poland*, pages 196–210, 2015.

[9] H. Friedman. Classically and intuitionistically provably recursive functions. In *Higher set theory (Proc. Conf., Math. Forschungsinst., Oberwolfach, 1977)*, volume 669 of *Lecture Notes in Math.*, pages 21–27. Springer, Berlin, 1978.

[10] J. Frey and J.G. Simonsen. Toposes for Time Complexity Classes, 2016. *Developments in Implicit Computational Complexity (DICE 2016)*, available at https://lipn.univ-paris13.fr/DICE2016/Abstracts/paper_6.pdf.

[11] J.Y. Girard. A new constructive logic: classic logic. *Mathematical Structures in Computer Science*, 1(03):255–296, 1991.

[12] J.Y. Girard. Locus solum: From the rules of logic to the logic of rules. *Mathematical Structures in Computer Science*, 11(03):301–506, 2001.

[13] T. Griffin. A formulae-as-type notion of control. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 47–58. ACM, 1990.

[14] J.M.E. Hyland, P.T. Johnstone, and A.M. Pitts. Tripos theory. *Math. Proc. Cambridge Philos. Soc.*, 88(2):205–231, 1980.

[15] J.M.E. Hyland and C.H.L. Ong. On full abstraction for PCF: I, II and III. *Inform. and Comput.*, 163(2):285–408, 2000.

[16] J.L. Krivine. A call-by-name lambda-calculus machine. *Higher-Order and Symbolic Computation*, 20(3):199–207, 2007.

[17] J.L. Krivine. Realizability in classical logic. *Panoramas et synthèses*, 27:197–229, 2009.

[18] Y. Lafont. Negation versus implication. *Logical Frameworks*, pages 223–229, 1991.

[19] Y. Lafont, B. Reus, and T. Streicher. Continuation semantics or expressing implication by negation. Unpublished, available at http://iml.univ-mrs.fr/~lafont/pub/continuation.ps, 1993.

[20] A. Miquel. Existential witness extraction in classical realizability and via a negative translation. *Log. Methods Comput. Sci.*, 7(2):2:2, 47, 2011.

[21] P.A. Melliès and N. Tabareau. Resource modalities in tensor logic. *Ann. Pure Appl. Logic*, 161(5):632–653, 2010.

[22] G.D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoret. Comput. Sci.*, 1(2):125–159, 1975.

[23] P. Selinger. Control categories and duality: on the categorical semantics of the lambda-mu calculus. *Math. Structures Comput. Sci.*, 11(2):207–260, 2001.

[24] T. Streicher and B. Reus. Classical logic, continuation semantics and abstract machines. *Journal of functional programming*, 8(06):543–572, 1998.

[25] K. Terui. Computational ludics. *Theor. Comput. Sci.*, 412(20):2048–2071, 2011.

[26] H. Thielecke. Continuation semantics and self-adjointness. *Electronic Notes in Theoretical Computer Science*, 6:348–364, 1997.

[27] I. Urbas. Dual-intuitionistic logic. *Notre Dame J. Formal Logic*, 37(3):440–451, 1996.

[28] N. Zeilberger. On the unity of duality. *Ann. Pure Appl. Logic*, 153(1-3):66–96, 2008.

A Boolean (pre)algebras and Boolean triposes

This appendix recalls the definitions of *Boolean (pre)algebra* and *strict Boolean tripos*, and states an auxiliary lemma to characterize Boolean prealgebras.

Definition A.1 A *Boolean algebra* is a complemented distributive lattice, i.e. a distributive lattice $(B, \leq, \top, \wedge, \bot, \vee)$ such that for every $a \in B$ there exists a $b \in B$ with $a \wedge b = \bot$ and $a \vee b = \top$.

A Boolean *prealgebra* is a preorder whose poset-reflection is a Boolean algebra.

The term 'Boolean prealgebra' does not seem to be very prevalent in the literature, but it appears e.g. in [6].

Lemma A.2 *A preorder $(B, \leq)$ is a Boolean prealgebra if and only if it has finite meets (denoted by $\wedge, \top$) and there exists a function $\neg(-) : B \to B$ such that*

$$\textit{(i)}\ \ a \wedge b \leq \neg\top \ \ \Rightarrow \ \ a \leq \neg b \qquad \textit{(ii)}\ \ a \wedge \neg a \leq \neg\top \qquad \textit{(iii)}\ \ \neg\neg a \leq a$$

for all $a, b \in B$.

Proof. The following derivation shows that $\neg(-)$ is antimonotone.

$$\dfrac{\dfrac{\dfrac{a \leq b}{\neg b \wedge a \leq \neg b \wedge b} \qquad \dfrac{}{\neg b \wedge b \leq \neg\top}}{\neg b \wedge a \leq \neg\top}}{\neg b \leq \neg a}$$

The converse implication of *(i)* is shown as follows.

$$\dfrac{\dfrac{\dfrac{a \leq \neg b}{a \wedge b \leq \neg b \wedge b} \qquad \dfrac{}{\neg b \wedge b \leq \neg\top}}{a \wedge b \leq \neg\top}}{}$$

The following shows that $\neg(-)$ is an involution,

$$\dfrac{a \wedge \neg a \leq \neg\top}{a \leq \neg\neg a}$$

which implies that $(A, \leq)$ is auto-dual and hence a lattice. The non-trivial direction

of distributivity is shown as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg(a\wedge b)\wedge a\wedge b\leq\neg\top}}{\neg(a\wedge b)\wedge a\leq\neg b}}{\neg(a\wedge b)\wedge a\wedge\neg c\leq\neg b\wedge\neg c}}{\neg(a\wedge b)\wedge a\wedge\neg c\leq\neg\neg(\neg b\wedge\neg c)}}{\neg(\neg b\wedge\neg c)\wedge\neg(a\wedge b)\wedge a\wedge\neg c\leq\neg\top}}{\neg(\neg b\wedge\neg c)\wedge\neg(a\wedge b)\wedge\neg c\leq\neg a}\qquad\dfrac{}{\neg(\neg b\wedge\neg c)\wedge\neg(a\wedge b)\wedge\neg c\leq\neg c}}{\neg(\neg b\wedge\neg c)\wedge\neg(a\wedge b)\wedge\neg c\leq\neg a\wedge\neg c}}{\neg(\neg b\wedge\neg c)\wedge\neg(a\wedge b)\wedge\neg c\leq\neg\neg(\neg a\wedge\neg c)}}{\neg(\neg a\wedge\neg c)\wedge\neg(\neg b\wedge\neg c)\wedge\neg(a\wedge b)\wedge\neg c\leq\neg\top}}{\neg(\neg a\wedge\neg c)\wedge\neg(\neg b\wedge\neg c)\leq\neg(\neg(a\wedge b)\wedge\neg c)}}{(a\vee c)\wedge(b\vee c)\leq(a\wedge b)\vee c}$$

It remains to check that for $a\in A$, $\neg a$ is a complement of a in the sense of the previous definition. This follows from *(ii)* and the fact that $\neg(-)$ is an involution.□

The following definition of *strict Boolean tripos* is a special case of the concept of tripos as introduced in [14].

Definition A.3 A *strict Boolean tripos* is a contravariant functor

$$\mathscr{P} : \mathbf{Set}^{\mathsf{op}} \to \mathbf{BA}$$

from the category of sets to the category of Boolean prealgebras and structure preserving maps such that

- for any $f : J \to I$, the map $\mathscr{P}(f)$ has a left [5] adjoint $\exists_f$ (which is not required to preserve Boolean prealgebra structure), such that for any pullback square

$$\begin{array}{ccc} L & \xrightarrow{q} & K \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle g} \\ J & \xrightarrow{f} & I \end{array}$$

  we have $\mathscr{P}(g)\circ\exists_f = \exists_q \circ \mathscr{P}(p)$ (this is the *Beck-Chevalley condition*), and
- there exists a *generic predicate*, i.e. a set Prop and an element $\mathsf{tr}\in\mathscr{P}(\mathsf{Prop})$ such that for every set I and $\varphi\in\mathscr{P}(I)$ there exists a unique $f : I\to\mathsf{Prop}$ with $\mathscr{P}(f)(\mathsf{tr})=\varphi$.

[5] Note that the right adjoint $\forall_f$ is for free in the Boolean case, it is given by $\forall_f\varphi = \neg\exists_f\neg\varphi$.